L7: Entry 13 of 17 File: PGPB Nov 11, 2004

DOCUMENT-IDENTIFIER: US 20040225680 A1
TITLE: Declarative rules for metadirectory

Current US Classification, US Primary Class/Subclass:
707/104.1

Detail Description Paragraph:
[0045] Each of the exemplary schemas 302 is a model for describing the structure of information processed by the metadirectory. The schemas 302 also include criteria specifying how data is processed in the metadirectory. In general, a separate schema may be developed and applied at any place within the metadirectory that information crosses a boundary. For example, with regard to the metadirectory 200 of FIG. 2, an import schema may be applied to the information 222 when the information 22 is received from the remote repository 224. As another example, a core schema may be applied to the information 222 when it is moved from the buffer 216 into the core 214. Schemas 302 may hold information such as system configuration information, data format specifications, and others.

Detail Description Paragraph:
[0056] Metadirectory Core Schema.

Detail Description Paragraph:
[0163] The optional core object type parameter is employed when the resolution script has the ability to determine updates to the object type of the core join target object. This ability might be beneficial in interforest-type scenarios where temporary placeholder objects (e.g., a "contact") may be initially imported into the core from a non-authoritative source, and it may be desirable to have the authoritative source override the object type when joining to it (e.g., change a "contact" to a "user").

Detail Description Paragraph:
[0164] When a resolution script indicates that the object type of the core object must change, the connector object in the buffer is joined to the core object. Subsequently, all attribute values contributed by the metadirectory buffer objects joined to a core object, will be recalled, the object type of the core object will be changed, and attributes will be re-imported from all the buffer objects joined to the core object. After doing so, provisioning and export flow processes are performed.

Detail Description Paragraph:
[0172] Declarative mappings allow a user to specify types of metadirectory core objects that should be created when a given remote repository object type is to be projected. Each declarative mapping is an object type pair that identifies both a source remote repository object type and a destination metadirectory core object type. Such declarative mappings may be one-to-one mappings or many-to-one mappings. An example of a many-to-one mapping is a mapping of both "employee" object types and "contact" object types to a "user" object type in the core. A many-to-one mapping may be accomplished by employing two distinct mappings (e.g., one mapping for "employee" to "user", and another for mapping "contact" to "user").

Detail Description Paragraph:
[0196] Flow and mapping sub-elements of the <export-flow-set> element define flows or relationships between a metadirectory buffer object type and metadirectory core object type pair. More specifically, the exemplary <export-flow-set> element shown above defines cd-object-type and mv-object-type attributes that serve to define the scope of sub-elements by source metadirectory core object type and destination metadirectory buffer object type. In an exemplary implementation, the value of the cd-object-type attribute is the name of a buffer object type defined in a destination MA's schema, and the value of the mv-object-type attribute is the name of an object type defined in the Core schema (described in further detail below). In one implementation, each flow set corresponds to one source core object type and one destination buffer object type. An exemplary <export-attribute-flow> element is shown below in XML:

Detail Description Paragraph:
[0288] Exemplary Metadirectory Core Schema

Detail Description Paragraph:
[0289] An exemplary metadirectory core schema is provided to describe aspects of the metadirectory core at a high-level. As with the exemplary MA schema, elements in the exemplary metadirectory core schema can be broken out into sub-elements. An exemplary metadirectory schema and exemplary sub-elements are described below:

Detail Description Paragraph:
[0293] An exemplary <timeout> tag contains an integer specifying a number of seconds for a timeout. If set to 0, the timeout is disabled. The <timeout> value may be a global setting, in which case, the timeout will be specified only in the Metadirectory Core schema.

Previous Doc Next Doc Go to Doc#

L7: Entry 16 of 17 File: USPT Feb 13, 2007

DOCUMENT-IDENTIFIER: US 7177843 B2
TITLE: Rights expression system

PRIOR-PUBLICATION:

DOC-ID	DATE
US 20030125976 A1	July 3, 2003

Brief Summary Text (14):
A second music company who allows downloads may define yet another extension, called "S2", to include a right called "download" for building expressions such as "any subscriber can download music content X0", assuming the grammar defines the condition to be optional. Note that S1 extends the original schema Core, enabling it to use any definitions provided by Core. Also, since S2 extends S1, S2 may use definitions provided by both S1 and Core. The above exemplary Core schema is described using XML Schema syntax in FIG. 13. As can be appreciated, the Core schema basically defines a grant that includes four different types: principal, right, resource and condition in the present example.

Description Paragraph (36):
As shown in TABLE 1, the REL terms are listed along with their respective type, stylesheet, prefix, and namespace. In the above example of the terms dictionary 16 as shown in TABLE 1, the core terms are listed under the "Term" heading. A term type indicates the term category, e.g., whether it's a "right", a "principal", or a "condition". Thus, as previously noted, if the "right" type is selected, REL terms corresponding to the Right type such as "view", "print" are presented to the user by the rights expression system 1. For example, a "condition" selection would query the terms dictionary for all available conditions for the user's selection. Of course, the extensible core types are not limited to these types, but instead, may include other types such as issuer type.

Description Paragraph (40):
In addition, in accordance with one embodiment of the rights expression system 1, the transformer 17 is invoked to convert the generated rights expressions into their textual description for display in the layout which serves as a neutral space for the display of the rights expressions to facilitate user's comprehension of the rights expression. Furthermore, in accordance with one embodiment of the present invention, the layouts 2 are preferably capable of embedding terms from new REL extensions so long as a schema service 22 associated with the new REL extension is provided. This capability is made possible because type selection components such as one or more type selectors 3 for each extensible core type will recognize any terms defined for their type. For example, to support a new REL extension that defines an REL term called PassportUser, a schema service 22 may be added to the rights expression system 1 that is responsible for telling the terms dictionary 16 that it has the PassportUser term of the type "Principal". When the type selection component of the type "Principal" is invoked, it will show PassportUser as one of the selections available to the user.

Description Paragraph (57):
FIG. 10 is a flowchart 80 setting forth the process for creating the terms

dictionary 16 and the transformer 17 of FIG. 2. While being loaded, the data engine 5 instantiates the schema manager 18, which in turn, executes the procedure of the flowchart 80 of FIG. 10. In step 81, the application profile interface 20 is invoked to detect the existence of the application profile 19, and reads it into memory of the application profile interface 20. The schema manager 18 then searches the directory for schema services, schema service for REL 21 and schema service for extension 22. As previously noted, each of the schema services 21 and 22 provide information for the REL or one of its extensions. In step 82, each schema service 21 and 22 is queried for terms derived from the core elements of the REL. In the rights expression system 1 of the present embodiment, any derivations of the following core types are accounted for: principal, resource, condition, right and issuer. Of course, in other embodiments, other core types may be accounted for as well.

Description Paragraph (60):
In step 94, a determination is made as to whether the supporting schema service is found, and the nature of the rights expression is determined, including what core type it is and what its supporting schema service is. If no supporting schema service is found, a determination is made as to whether more rights expressions exist to process in step 97. If more rights expressions need to be processed, the process reverts to step 92 to iteratively process the rights expressions until no more rights expressions need to be processed so the process ends in step 98.

Current US Cross Reference Classification (11):
707/104.1

Current US Cross Reference Classification (12):
707/9

Previous Doc Next Doc Go to Doc#

L7: Entry 17 of 17 File: USPT Mar 9, 2004

DOCUMENT-IDENTIFIER: US 6704743 B1
TITLE: Selective inheritance of object parameters in object-oriented computer environment

Brief Summary Text (10):
One aspect of a computer language that weighs heavily on the expressiveness and intuitiveness of the language is the number and types of core semantic elements used by the language. In particular, despite the wide differences in computer languages, at the very essence, every computer language relies on a set of core semantic entities that represent the basic "words" for the language. Just as the words in a dictionary form the core elements for the English language from which all communication in the English language is derived, the core semantic entities for a computer language are the basic elements from which all programmatic concepts are constructed.

Detailed Description Text (130):
Another important aspect of the illustrated entity management system is the concept of database seeding. While some implementations may require no database seeding, and may rely on the aforementioned object classifications, the illustrated embodiment relies on some degree of seeding of basic entities to support a number of the types of unique functionality discussed herein, e.g., object inheritance (including implicit inheritance and selective inheritance), and bidirectional relationships (discussed below), among others. For each of these functionalities, feature definition entities may need to be created and placed in the database, including, for example, a "parent" relationship, a "should inherit" attribute, a "back link" relationship, etc. By supporting such functionalities in populated entities, rather than in the core schema, the core schema is kept as simple and lean as possible. However, in other implementations, such functions may be implemented directly within the core schema itself, rather than in any populated entities. Moreover, other functionalities described herein as being implemented within the core schema may be implemented in higher-level populated entities in some implementations.

Detailed Description Text (131):
FIG. 61, for example, shows how database seeding may be relied upon to support dynamic, user-definable object inheritance. The core schema described herein in not configured to include at its most basic level the concept of object inheritance. In the illustrated implementation, this concept is introduced by creating a parent definition entity 850 in the database, used to define the characteristics of a parent relationship 854 between any parent and child entity (e.g., parent entity 856 and child entity 858). This parent definition entity is typically referred to as a system definition, to distinguish it from user-created definitions.

Current US Original Classification (1):
707/103R

Current US Cross Reference Classification (1):
707/201

Current US Cross Reference Classification (2):
707/206

Previous Doc Next Doc Go to Doc#